Student Strategy Prediction using a Neuro-Symbolic Approach

Anup Shakya
University of Memphis
ashakya@memphis.edu

Vasile Rus
University of Memphis
vrus@memphis.edu

Deepak Venugopal
University of Memphis
dvngopal@memphis.edu

ABSTRACT

Predicting student problem-solving strategies is a complex problem but one that can significantly impact automated instruction systems since they can adapt or personalize the system to suit the learner. While for small datasets, learning experts may be able to manually analyze data to infer student strategies, for large datasets, this approach is infeasible. We develop a Machine Learning model to predict strategies from student data. While Deep Neural Network (DNN) based methods such as LSTMs can be applied for this task, they often have long convergence times for large datasets and like several other DNN-based methods have the inherent problem of overfitting the data. To address these issues, we develop a Neuro-symbolic approach for strategy prediction, namely a model that combines strengths of symbolic AI (that can encode domain knowledge) with DNNs. Specifically, we encode relationships in the data using Markov Logic and use symmetries among these relationships to train an LSTM more efficiently. In particular, we use an importance sampling approach where we sample the training data such that for clusters/groups of symmetrical instances (instances where the strategies are likely to be symmetric), we only pick representative samples for training the model instead of using the whole group. Further, since some groups may contain more diverse strategies than the others, we adapt the importance weights based on previously observed samples. Through empirical evaluation on the KDD EDM challenge datasets, we show the scalability of our approach.

Keywords

Intelligent Tutoring Systems, Learning Strategies, Neuro-Symbolic AI, Markov Logic Networks, LSTMs

1. INTRODUCTION

Intelligent Tutoring Systems (ITSs) [31] and more broadly adaptive instructional systems (AISs)[1] help a diverse population of students by adapting instruction to each learner thus accounting for different learning abilities, learning styles and education goals. Such adaptation leads to more engaging and effective learning. However, in order to build effective ITSs, it is important to understand how students learn and what learning and instructional strategies are most effective for whom and under what conditions. Specifically, students can follow several different *strategies* to learn the same content. For example, consider a simple math problem about solving a linear system of equations where $x+y+z = 9$ $x = y$ and $y = z$. One strategy is to perform systematic substitutions till there is an equation in terms of one variable and simply solve for that variable. However, another strategy could be to use transitivity to see that all three variables have the same value and use this to solve the problem. Depending upon the way a student thinks, one strategy could be easier or harder to grasp compared to the other. Thus, understanding the various ways in which students approach an instructional task will not only further our understanding of how learners learn, e.g., it may help identify the most effective learning strategies employed by top performers, but it will also enable ITSs to incorporate knowledge about these strategies in order to adapt appropriately and help students maximize their learning gains.

A student's choice of strategy is a complex function dependent on many factors such as experience with similar problems, general expertise in the topic, other cognitive abilities, etc. Human experts are exploring all these factors and how they are related to strategy use and learning. However, human experts are expensive and limited in the ability to analyze large data from thousands, tens of thousands, or millions of students. Advanced data science methods and access to large computing infrastructure such as the cloud offer new possibilities to analyze in-depth and at scale such large learner datasets with the promise of helping us discover, document, and benefit from the diversity of learning.

Indeed, with the growth of both data and advanced Machine Learning methods such as Deep Neural Networks (DNNs), we are able to successfully solve several challenging problems in domains such as natural language understanding [25] and visual processing [8]. However, the ability of DNNs to learn complex functions and representations comes at a cost. Specifically, DNNs require significant computational

[1]The main difference between Intelligent Tutoring Systems and Adaptive Instructional Systems at least in our view is that the former offer full-adaptivity, i.e., both micro- and macro-adaptivity, whereas the latter can offer any type, e.g., just macro-adaptivity.

Anup Shakya, Vasile Rus and Deepak Venugopal "Student Strategy Prediction using a Neuro-Symbolic Approach". 2021. In: Proceedings of The 14th International Conference on Educational Data Mining (EDM21). International Educational Data Mining Society, 118-129. https://educationaldatamining.org/edm2021/
EDM '21 June 29 - July 02 2021, Paris, France

resources to scale up to large datasets and at the same time, as data increases, they may not always yield expected results since they have a tendency to *overfit* the data. That is, they work well on the data on which they were trained but their generalization performance on unseen data severely degrades. A paradigm that is gaining significant attention in the AI/Machine Learning research community is Neuro-symbolic AI [7] where we augment DNNs with symbolic models to regularize the DNN. This helps in improving both scalability and generalization by allowing DNNs to learn from smaller datasets with higher accuracy. In this paper, we apply a Neuro-symbolic model to predict student strategies from structured student-interaction data.

Our model works on student data where the student interacts with an ITS in discrete steps. The strategy prediction task in this case can be formulated as a sequence learning problem where we want to learn to predict the sequence of steps a student is likely to follow for a given problem. Knowing this sequence will provide the ITS prior information that it can use to adapt, e.g., by better tailoring the available hints and feedback. Sequence learning is applicable to many tasks in general with one of the most popular applications being machine translation where the goal is to translate a sequence of words from one language to another language. A widely used traditional approach that has been applied in sequence learning is Hidden Markov Models (HMMs). However, this assumes a one-step dependency where each step depends only upon the prior step. Student strategies in problem solving though are much more complex where a student action in one step can influence several downstream steps. Long Short-Term Memory (LSTMs) [11] are DNNs that can model such long-term dependencies in sequences. However, LSTMs are known to take an extremely long time to train for large datasets [39]. To address this, we propose a Neuro-symbolic model where we combine the semantics of a symbolic AI model called Markov Logic [9] with LSTMs. Markov Logic encodes domain-knowledge using first-order logic formulas. The formulas establish relationships in the data that can be described in the form of a graph structure. Our approach learns symmetries between instances based on the graph structure and then uses these symmetries to train an LSTM more efficiently. Specifically, we use *importance sampling* to choose a subset of training instances to make learning more efficient. While importance sampling-based learning in DNNs has been used previously to scale up training [1, 13, 14], most existing approaches determine the importance of a training instance by estimating the gradient norm which is computationally expensive [14]. In our case, we determine importance of a training data instance based on symmetries in the MLN graph. Specifically, the idea is that if several strategies are likely to be symmetrical then we can learn more efficiently from a smaller subset of strategies instead of the whole training set. To do this, we learn an embedding such that problem instances which have symmetries in the graph have similar vector representations in the embedding. We then cluster the embedding vectors and sample instances from each cluster to train the model. The idea is to sample a subset with same overall distribution of strategies as the original dataset. That is, we end up with a smaller dataset the preserves much if not all the information in the original dataset. However, since the clustering is approximate, we may end up with some clusters where the strategies are likely to be more diverse than others. Therefore, we adaptively train the model by updating importance weights for the clusters in iteration $t+1$ based on the trained model in iteration t. Specifically, we sample more data instances from a cluster in $t+1$ when the model trained in iteration t has smaller accuracy for instances sampled from that cluster.

We evaluate our approach on the publicly available KDD EDM challenge datasets [34]. We compare our approach with HMMs and pure LSTM methods that do not use symmetries in training the data and show that our proposed Neuro-symbolic model is more accurate and scalable, where we obtain high prediction accuracy by focusing on a small fraction of the training data.

2. RELATED WORK

Ritter et al. [30] provide a comprehensive survey on different approaches used to identify student strategies. Model tracing based tutors [4] have been previously used to identify strategies. In such cases, strategies may be pre-specified and the tutor can recognize correct and incorrect strategies. Model-tracing based methods have also been adapted to recognize new strategies [29]. Sequence learning has been widely used for strategy identification. Specifically, in Open Ended Learning Environments such as Betty's brain [23], student activities were captured in logs and sequence pattern mining methods was used on these logs to extract action sequences which in principle are similar to sequences that we consider in this paper. Different types of strategies based on these sequences were analyzed in multiple studies [16, 17, 18] which also mapped these sequences to performances to compare and analyze strategies followed by high performers to those followed by low performers. Sequence learning has also been used to extract strategies for self-regulated learning [3]. More recently, a study performed large-scale sequence pattern mining in MOOCs platform to analyze activity sequences of learners [37]. Further, in the context of conversational tutors, tutorial dialogues can be treated as sequence of actions based on language-as-action theory [2, 33]. These sequences which are akin to strategies are mapped into a taxonomy by education experts [26]. Approaches have been developed to recognize these sequences from natural language interactions to help automated tutors understand successful strategies to guide a student. In particular, sequence learning methods have been used for this task as well [32, 24]. Symbolic models such as Markov Logic have also been applied for recognizing these sequences using joint inference [36]. In general, Neuro-symbolic models have gained prominence recently and have found applications in problems that have graph structure. In [22], the authors provide a detailed survey of Neuro-symbolic models using graph neural networks. In complex problems such as visual question answering that require connections between language and image processing, Neuro-symbolic models have performed better than pure neural network based methods [38]. Our proposed application in this paper is further validation that Neuro-symbolic AI is a promising direction to solve complex problems.

3. SEQUENCE LEARNING MODELS

Student strategies can be defined in different ways. In particular, the definition of what constitutes a strategy also

depends upon the type of interaction the student has with the AIS. In our case, we only consider structured interactions with discrete steps. Therefore, we define the student strategy as a function of the sequence of steps in the interaction. Each step is characterized by the central concept that is utilized to solve that specific step, i.e., the knowledge component [20] (KC) used in that step. Therefore, we define strategy in our case as a sequence of KCs used by a student in a problem solving session. Note that, this formulation of strategy as a sequence of discrete components is similar to the definitions used in prior work [30]. Formally,

DEFINITION 1. *Given a student s and a problem p, we define the strategy as $\bar{\mathbf{x}}_{s,p} = K_{s,p}^{(1)} \ldots K_{s,p}^{(n)}$, where $K_{s,p}^{(i)}$ is the knowledge-component that s uses to solve the i-th discrete step in p.*

We can now formulate a learning problem as follows. Given training data, $\{\mathbf{x}_{s_i,p_j}\}_{i=1,j=1}^{n,m_i}$, where n is the number of students and m_i is the number of problems solved by the i-th student, we learn a model $\mathcal{P}$ such that for a student s' and problem p', $\mathcal{P}$ generates a sequence of knowledge components $K_{s',p'}^{(1)} \ldots K_{s',p'}^{(k)}$. Note that students sometimes use more than one KC per step, in this case, we just unroll these multiple KCs by repeating the step with each of the KCs. Therefore, for the rest of this paper, we treat both multiple KCs in a step and single KC steps without distinguishing them. Also, to keep notation simpler, instead of adding the subscripts s_i, p_j each time, we denote the training input and output pairs as $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{N}$.

3.1 HMM Model

A popular model that is often used to learn from sequential data is the Hidden Markov Model (HMM), where we assume that every time-step is dependent on the previous time-step. HMMs are generative models represented using a dynamic Bayesian network. Each step in the time series is encoded by a *hidden-state* variable in the Bayesian network. We connect the hidden node corresponding to time-step j to the hidden node $j+1$ in the network and the observed feature at step j is also connected to the hidden node at step j. In our case, we encode the HMM as follows. Let O_t be the random variable representing the knowledge component at step t. We encode the knowledge component at step t as a vector $\mathbf{O}_t$ (the value of the random variable O_t) using a one-hot encoding. Let Z_t be the hidden-state variable corresponding to step t. The emission probability is given by $P(\mathbf{O}_t|Z_t)$ and the transition probability is given by $P(Z_t|Z_{t-1})$, i.e., the hidden state at time t depends only upon the hidden state at time $t-1$. The transition matrix is a $k \times k$ matrix that specifies transition probabilities to a state at time t given any other state at time $t-1$. Here, k which specifies the number of hidden states is pre-defined in the model.

The learning task in the HMM is to estimate the parameters of the HMM, namely, the transition matrix and the parameters of the emission probability distributions. Note that we need to estimate conditional probability conditioned on each possible state of the latent variable. To do this, we assume a Gaussian distribution represents the probability of each hidden state and the emission probabilities are also Gaussian distributions. Using the EM (Expectation Maximization) algorithm, we compute the parameters of the distributions using Max-likelihood estimation which has guarantees on convergence to a local optima. For predicting the strategy, we sample a KC at the first time step. Then, given a KC at any time step t, we generate the KC at time step $t+1$ as follows. Based on the observation, i.e., KC at step t, we predict an equivalent hidden state representation at step t and using the transition probability matrix, we sample the hidden state at time step $t+1$. We then predict the KC at time step $t+1$ using the emission probability.

3.2 LSTM Model

One of the problems with HMMs is that they have restrictive assumptions, i.e., each step depends only on the previous step. Ideally we would like to consider the student's activity across several steps to determine what his/her next step in the strategy is likely to be. For instance, suppose we have a student who works out a problem using a *divide-and-conquer* strategy, then there may be several small subproblems that the student solves before combining them together. In this case, a HMM model that simply looks at the previous step performed by a student may be able to capture the local strategy but will typically be unable to infer the global strategy since the dependencies may run across several steps. Therefore, to infer such advanced strategies, we need a more sophisticated model that captures longer-term dependencies. Long Short Term Memory (LSTMs) [11] are a variant of recurrent neural networks that have been used successfully for several problems like modeling text data. In particular, LSTMs can exploit longer range dependencies across words/sentences to learn a latent representation of sentences/documents. In our case, we apply LSTMs to learn a latent representation of the strategy.

Unlike HMMs, LSTMs are discriminative models that predict an output for step t based on the features observed in step t as well as a hidden state vector that summarizes the information up to step $t-1$. Note that a bi-directional LSTM can also consider information in steps succeeding t. To learn an LSTM for our task, we construct a tensor $T \in \mathbb{R}^{m \times n \times k}$, where m is the number of training instances, n is the number of steps and k is the dimensionality of features representing each training instance (s, p). Note that we can represent variable-length strategies using a special Start and Stop symbol in the LSTM to denote the start and end of strategies. The output of the LSTM for the t-th step is a vector representing the KC at step t. The final hidden state vector summarizes information for the full strategy for an input instance.

4. NEURO-SYMBOLIC MODEL

Though an LSTM can be directly used to learn a model for strategy prediction, it has certain limitations. LSTMs are known to converge very slowly for large datasets [39]. Further, LSTMs treat each instance in the training data as i.i.d (independent and identically distributed) which is limiting when there are underlying relationships among the instances. For example, problems are related to each other if solved by the same student, KCs used by the same student in similar problems are related to each other, etc. We address these limitations combining LSTMs with a symbolic model.

Neuro-symbolic AI [7], namely, combining symbolic AI mod-

els with DNNs has gained significant attention in tasks such as Visual Question Answering [38]. Neuro-symbolic models augment deep learning with knowledge from symbolic models. This can help learn deep models more efficiently even with limited labeled data [35]. Further, augmenting DNNs with symbolic models also controls overfitting. Specifically, since DNNs are highly expressive models, they are known to sometimes overfit the training data, particularly when the training data is non-diverse and in many problems such as image classification, data augmentation methods [5] seek to increase data diversity. In our Neuro-symbolic approach, we represent relationships in our dataset using the language of Markov Logic [9]. Based on symmetries in the relationships, we sample a smaller, more diverse training dataset to train the LSTM efficiently.

4.1 Markov Logic

Markov Logic [9] is a symbolic AI model designed as a representation and reasoning language for relational data. Markov Logic specifies relationships using the language of first-order logic. Each formula in Markov logic specifies a logical relationship using variables which can be substituted by symbols (also called constants or objects) from the data.

For example, we can model the fact that if two problems are similar then they require the same KC as a formula such as $\texttt{KC}(p_1, k) \wedge \texttt{Similar}(p_1, p_2) \Rightarrow \texttt{KC}(p_2, k)$. We can substitute this general formula using symbols in the data, say two problems P_1 and P_2 and KC K to obtain the formula, $\texttt{KC}(P_1, K) \wedge \texttt{Similar}(P_1, P_2) \Rightarrow \texttt{KC}(P_2, K)$. The formula that is substituted with the symbols is called a *ground-formula* using terminology from first-order logic. The predicates that are substituted with symbols are called *ground-atoms*, e.g., $\texttt{KC}(P_1, K)$ is a ground atom of predicate KC.

A Markov Logic Network (MLN) can be represented as a graph where the relationship encoded in each ground formula is represented by a clique in the graph. A clique is *activated* if the logical relationship specified by the ground-formula corresponding to that clique is evaluated to True in the data. For instance, in the above example, the clique corresponding to KC (P_1, K), Similar (P_1, P_2) and KC (P_2, K) is activated if the data asserts the logical relation that problems P_1 and P_2 use KC K and problem P_1 is similar to P_2. In MLN semantics, each activated clique represents a function parameterized by a weight attached to the ground formula. The full graph represents an undirected probabilistic graphical model [21]. For large datasets in real-world applications such as ours, the number of ground-formulas become very large resulting in an extremely large graph. In the standard use of MLNs, we learn parameters for the MLN based on Max-Likelihood Estimation (MLE). However, computing the gradient for MLE is infeasible when the graph is constructed from large datasets such as ours. Note that though, parameters for the MLN are required only if we want to use the MLN directly for probabilistic inference, i.e., when we want to answer queries using the MLN. While this is certainly desirable, it is well-known that MLN inference/learning algorithms cannot scale up to large datasets and perform extremely poorly in such cases [15]. Therefore, in our case, we make a simplifying assumption in the MLN that all the parameters for the graphical model have uniform weights. Thus, in our Neuro-symbolic model, Markov Logic is only used as a *language for knowledge representation (KR)* and not for inference/learning. That is, formulas specify relationships/connections in the MLN graph between different entities in our dataset (e.g. problems, students, etc.) while the actual learning and predictions are performed by an LSTM model. Note that in theory, other forms of KR such as Bayesian networks, arithmetic circuits or probabilistic programs can be used. However, the benefit of MLNs is that they specify relationships over large data using compact first-order formulas. Next, we describe the formulas in our MLN followed by how we learn symmetries in the graph to train the LSTM more efficiently.

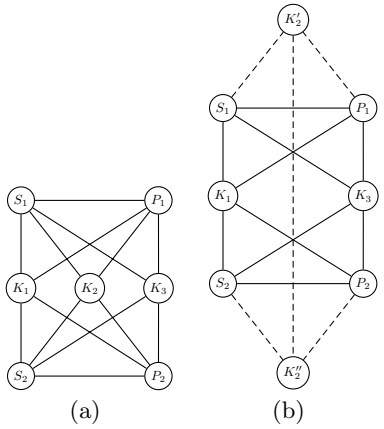

Figure 1: Illustrating symmetries in the MLN object graph. The graph represents symbols/objects in the MLN and the edges represent connection between variables, i.e., if objects appear together in a formula, they are connected in the graph. In (a) student S_1 can be exchanged with student S_2 and problem P_1 with P_2 to get an isomorphic graph. In (b) if the KCs K_1 and K_2 are similar, then the exchange is approximate.

4.1.1 MLN Structure

Our first set of MLN formulas relates the KCs to the problem and the student solving the problem.

$$\texttt{Student}(s) \wedge \texttt{Problem}(p) \wedge \texttt{PHierarchy}(p, h) \Rightarrow \texttt{KC}(s, p, t, k)$$

where s is a variable that denotes a student, p denotes a problem, t is a step, k denotes a knowledge component and h denotes the problem hierarchy which is the hierarchy of curriculum levels containing the problem, and PHierarchy(p,h) relates to the problem p in the hierarchy h where the hierarchy contains the curriculum unit name and the section name that the problem belongs to (e.g. Unit LCM, Section LCM-2).

Next, we encode the *homophily property* where the same KC is likely to be reused by a student for problems that are related to each other through a common problem hierarchy.

$$\begin{aligned}\texttt{Student}(s) \wedge \texttt{Problem}(p_1) \wedge \texttt{Problem}(p_2) \wedge \\ \texttt{PHierarchy}(p_1, h) \wedge \texttt{PHierarchy}(p_2, h) \wedge \texttt{KC}(s, p_1, t_1, k) \\ \Rightarrow \texttt{KC}(s, p_2, t_2, k)\end{aligned}$$

Next, we encode transitive dependencies between KCs by relating KCs that occur close to each other. Specifically, this encodes local dependencies across KCs (similar to a HMM model).

$$\texttt{KC}(s, p, t-1, k_1) \wedge \texttt{KC}(s, p, t, k_2) \Rightarrow \texttt{KC}(s, p, t+1, k_3)$$

4.2 Embeddings

Given the formulas, we can define the MLN object graph by connecting objects that appear in the same ground formula. For instance, consider an example MLN object graph shown in Fig. 1 (a) with two students (S_1, S_2), two problems (P_1, P_2) and three knowledge components (K_1, K_2, K_3). The edges indicate that in the graph corresponding to the MLN, there is a connection that relates the corresponding symbols. Note that S_1 works on P_1 and S_2 works on P_2, where P_1 and P_2 are related since they correspond to the same topic. Suppose both $S1$ and S_2 use the same strategy, K_1, K_2, K_3, then, we can exchange S_1 with S_2 and P_1 with P_2 to get an isomorphic graph structure. Now, suppose $S2$ uses a strategy K_1, K_2', K_3 that is slightly different from the strategy of S_1, say, K_1, K_2'', K_3, then we obtain a graph structure shown in Fig. 1 (b), where the new connections are shown by dotted lines. Now, exchanging S_1 with S_2 and P_1 with P_2 will not give us a graph structure that is isomorphic to the original graph. However, suppose that the knowledge components K_2' and K_2'' are similar to each other, i.e., there are many other problem instances where students use the KCs K_2' and K_2'' interchangeably, then exchanging S_1, P_1 with S_2, P_2 will yield an approximation that is still quite similar to the original graph. This means that using (S_1, P_1), it is reasonable to obtain a model that can predict the strategy for (S_2, P_2) and vice-versa. The set $\{(S_1, P_1), (S_2, P_2)\}$ is therefore an equivalence class consisting of approximately symmetrical instances. In general, if we group together approximately symmetrical instances in our training data, then we can train our LSTM model with diverse instances by sampling these groups since each group represents data that is likely to have a similar effect in training the model. We do this by learning *embeddings* for nodes in the graph structure.

Unfortunately, the size of the graph becomes very large as we increase dataset size and it is practically infeasible to construct the graph structure explicitly. Though several approaches have been developed that identify symmetries in MLNs using graph automorphism groups [27] using tools such as saucy [6], these approaches generally work directly on the graph structure. Further, it is possible to infer symmetries in graphs using other neural-network based methods such as Node2Vec [10] and Graph Convolutional Networks (GCNs) [19]. However, all these approaches work on general graphs and considering an MLN graph as a general graph is problematic since the graph becomes very large even for smallish datasets. This makes it hard to apply such approaches to our strategy identification problem since we expect to have an extremely large graph. Therefore, we instead use a recent, much more scalable Markov Logic graph specific approach called Obj2vec [12] that detects symmetries without explicitly constructing the graph. This approach is based on identifying approximate symmetries based on neighborhoods of a node using a neural network without constructing the actual graph.

4.2.1 Obj2Vec

Obj2Vec is inspired by skip-gram models [25] which are widely used to learn word embeddings. In skip-gram models, we learn an embedding for a word based on its *context*, i.e., the neighboring words that it typically appears with in text documents. For words which have similar contexts, we learn similar vector representations. Word2vec [25] is arguably the most popular skip-gram model, where we train a neural network for learning the embedding. Specifically, for each word w as input, the neural network learns to predict the context of w. Typically, The inputs and context words are encoded as one-hot vectors. The hidden layer in the neural network typically has a much smaller number of dimensions as compared to the input/output layers. The hidden-layer learns a dense, low-dimensional embedding, where similar words have similar vector representations. This is because, words that are similar typically have similar contexts in text documents and therefore the neural network learns a similar representation in the hidden layer for such words.

Obj2Vec extends the idea of word embeddings to MLNs. Specifically, recall that each ground formula of an MLN represents a clique in the MLN graph. For each activated ground formula, i.e., formula that represents a relationship that is supported by the data, we predict a symbol/object in the formula from other symbols/objects in that same formula. For example, suppose our data shows that *Alice* and *Bob* use the knowledge component `Slope-Intercept` across several problems. Then, all ground formulas that contain either *Alice* or *Bob* and the KC `Slope-Intercept` are activated. In this case, both *Alice* and *Bob* are said to share a common context. Therefore, we predict the symbol `Slope-Intercept` from both *Alice* and *Bob*. That is, we have an autoencoder neural network where the input is a one-hot encoding of *Alice* (or *Bob*) and the output is a one-hot encoding of `Slope-Intercept`. The neural network must therefore learn a common representation for both *Alice* and *Bob* since it needs to make similar predictions for both. Thus, suppose the hidden-vector representation (or embedding) for *Alice* is v_{Alice} and that for *Bob* is v_{Bob}, then $v_{Alice} \approx v_{Bob}$. Note that the embedding defines a continuous approximation of symmetries in relationships specified in the MLN graph. That is, the distance between the vectors v_{Alice} and v_{Bob} quantifies the symmetry between *Alice* and *Bob* based on relationships specified in the data.

4.3 Scalable Learning using Symmetries

The embedding vectors from Obj2Vec encodes relational knowledge from the MLN graph. Given the embedding vectors, we now train an LSTM to predict the student's strategy. Specifically, let our input instances be $\mathbf{x}_1 \ldots \mathbf{x}_N$, where each $\mathbf{x}_i$ consists of embeddings for a specific student s solving problem p, and the outputs are $\mathbf{y}_1 \ldots \mathbf{y}_N$, where $\mathbf{y}_i$ is a sequence of KCs used by the student s to solve the problem p. The LSTM training objective is given by,

$$\theta^* = \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(\psi(\mathbf{x}_i, \theta), \mathbf{y}_i) \quad (1)$$

where θ^* and θ represent the parameters of the LSTM, $\mathcal{L}$ is a loss function and $\psi(\mathbf{x}_i, \theta)$ is the sequence of KCs output

by the LSTM parameterized by θ for input $\mathbf{x}_i$. In general, a stochastic gradient descent (SGD) procedure can be used to minimize the objective in Eq. (1). In SGD, we sample the training instances to approximate the gradient. Typically, SGD assumes that all training instances are equally important, and therefore samples them uniformly. That is, the probability of sampling a specific instance in the training data is equal to $p = \frac{1}{N}$. However, this approach is expensive particularly if we repeatedly choose training instances that are similar to each other. For example, suppose all the training instances that we sample are likely to encode similar strategies, then our model may take a long time to understand diverse strategies. Further, since the underlying data encodes symmetries (from the MLN graph), the information in one training instance may be very similar to the information in another symmetrical instance. Therefore, we force the model to learn from instances with diverse relationships by imposing an importance distribution over the training data. Specifically, training instances with larger importance are more likely to be chosen as compared to training instances with smaller importance.

In general, to focus the training on more important data instances, we can modify the sampling distribution such that each instance is sampled with a non-uniform probability. This approach has been explored in prior work, where we scale up training by replacing the uniform distribution over the training instances with an importance distribution that quantifies how important a specific example is for the training process [14]. Previous work such as [14, 1, 13], have focused mainly on approximating importance as a function of the gradient norm which is hard to compute exactly. In [14], therefore, the authors propose an approximation to the gradient norm and use this to target important training examples. The focus in these approaches is to target the training examples that are likely to induce changes when updating the model parameters during backpropagation which can be shown to translate to a reduced variance in the gradient estimates. However, in our case, we have more information apriori in the embeddings to identify importance of a training example in terms of their relationships. Specifically, recall that the embeddings are based on symmetries in the MLN-graph which encodes relational knowledge. Thus, if two embeddings are similar, then it means that they share similar relationships. For example, if two student embeddings are similar, then it is likely that for the problems both students have solved, their strategies use similar KCs. Thus, using embedding-similarities, our model focuses the training effort on instances that encode diverse relationships.

4.3.1 Adaptive Importance Weighting

Given the instances $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{N}$, we cluster the instances using K-Means clustering. Each instance internally has two components, the student embedding as well as the problem embedding. Since we want to exploit symmetries in both, we cluster them along both dimensions. Let $\{\mathbf{C}_s^{(i)}\}_{i=1}^{n_1}$ and $\{\mathbf{C}_p^{(i)}\}_{i=1}^{n_2}$ denote the clusters found by K-Means using the student embeddings and the problem embeddings respectively, where n_1 and n_2 are the number of clusters. We now sample from each cluster to obtain a reduced set of training examples. For each cluster, we assign an importance weight to quantify how often we need to sample that cluster. Let q_i represent the importance weight of the i-th

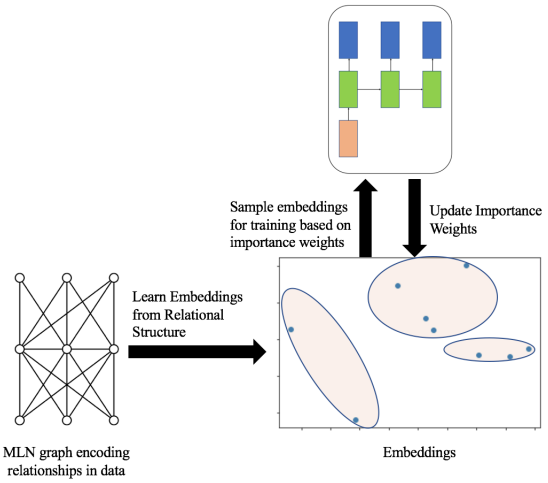

Figure 2: Schematic illustration of our proposed approach. We learn embeddings based on symmetries in the relational data. We then learn a one-to-many LSTM to map an input instance to a sequence that represents the strategy by focusing the training on a sample of the data that reflects symmetries in the data.

cluster. Ideally, we want to quantify q_i based on the symmetries encoded within the cluster. Specifically, if the cluster contains highly symmetric instances, then we require fewer training examples from the cluster. On the other hand, if the cluster contains diverse instances, we require more training samples from that cluster. One way to quantify this is using traditional clustering metrics such as within-cluster sum of squared errors (SSE) to measure cohesion within the cluster. That is, if the embedding vectors are close to each other within a cluster which implies a small SSE, then it is indicative that we may need fewer samples from that cluster. However, this approach may not necessarily yield the best results since both the embeddings and the clustering are simply approximations to true symmetries in strategies. For example, suppose the embedding vector for *Alice* is close to the vector for *Bob*, this means that considering all problems, *Alice* and *Bob* are approximately symmetrical. Similarly, if the vector for problem P_1 is close to problem P_2, this means considering all students, P_1 and P_2 are approximately similar. However, this may not always necessarily imply that the strategy followed by *Alice* for problem P_1 is guaranteed to be symmetrical to the strategy followed by *Bob* for P_2. Therefore, we use an adaptive approach to progressively learn these symmetries.

For training the model, from each cluster, we may require varying number of samples. That is, from clusters representing less symmetrical instances, we may require more samples while from other clusters that contain more symmetrical instances, we may require fewer samples. To account for this, we adapt the sampling as follows. We define the initial importance weight for the i-th cluster as $q_i^{(0)} = \frac{1}{K}$, where K is the number of clusters. Let $\theta^{(j)}$ be the parameters learned by the LSTM in iteration using samples from the clusters in iteration j. In iteration $j+1$, we update the importance

weight for the clusters as,

$$q_i^{(j+1)} = \frac{1}{m}\sum_{k=1}^{m}\mathcal{L}(\psi(\mathbf{x}_k^{(i)}, \theta^{(j)}), \mathbf{y}_k^{(i)}) \quad (2)$$

where $\mathbf{x}_k^{(i)}$ is a training example that consists of a randomly sampled instance from the i-th cluster. For example, if we are updating the i-th student cluster, $\mathbf{x}_k^{(i)}$ is a randomly sampled student s from this cluster combined with a problem p, where p is a problem that has been solved by s. Thus, if the LSTM in iteration j effectively encodes the symmetries in the i-th cluster, then the loss in Eq. (2) is likely to be small. Thus, $q_i^{(j+1)}$ is small. On the other hand, if $q_i^{(j+1)}$ is large, then we need more samples from the i-th cluster since the LSTM trained using the samples collected until iteration j does not effectively model the instances in the i-th cluster. The importance weights for the i-th cluster adaptively change from $q_i^{(0)}$ $q_i^{(1)}$ ... $q_i^{(T)}$. Note that each iteration adds training data to the LSTM and thus effectively increases training time. However, we do not begin the training from scratch in each iteration. Specifically, for iteration $j+1$, we consider the LSTM learned until iteration j as the starting point. This is similar to *pre-training* that is common in deep network training. For iteration $j+1$, the $\theta^{(j)}$ represents the pre-trained parameters of the LSTM. We stop adapting the weights after a fixed number of iterations depending based on a cutoff for the training time. Note that more advanced convergence criteria can be explored here which is a part of our future work. To summarize, Fig. 2 shows a schematic representation of our overall model.

5. EXPERIMENTS

5.1 Setup

We evaluate our approach on the publicly available KDD EDM challenge datasets, `Algebra 2008, 2009` and `Bridge to Algebra 2008, 2009` [34] which consists of data collected from the Mathia platform. Each instance consists of several discrete steps and each step is mapped to a knowledge component which is used to solve that step. The statistics for the two datasets are shown in Table 1. As shown in the table, these datasets are quite large with over 850K and 1.6M instances respectively. All our experiments were performed on a 64GB memory machine with a Nvidia GPU and an Intel Core-I9 processor. For computing accuracy, in each input instance, we compute the percentage of total steps where the true KC matches with the predicted KC. The overall accuracy is computed as the average accuracy across all instances. To measure variance of our estimates, for each of the results shown, we run the experiments 10 times and compute the mean accuracy and the standard deviation of the accuracy. Next, we describe the implementation of different approaches that we use in our experiments. The code and data for our implementations are available here[2].

5.1.1 Hidden Markov Model

We trained a Gaussian Hidden Markov Model (we refer to this as `HMM`) using the sklearn package in python. We set the number of hidden states to 100 and initialized the Gaussian emission probabilities with a full covariance matrix so that it has flexibility to generate varied sequences. We trained the model using the EM algorithm.

[2]https://github.com/anupshakya07/SSPM

Dataset	Total Instances	No. of Students	No. of Problems	No. of unique KCs
algebra_2008_2009	838728	3310	188368	541
bridge_to_algebra_2008_2009	1624951	6043	52754	933

Table 1: Details of the dataset.

Learning-rate	Optimizer	Batch-size	Dropout-rate	LSTM (hidden state)	Obj2Vec embedding
0.001	Adam with CCE-Loss	100	0.3	200 dimensions	300 dimensions

Table 2: Parameters for training.

5.1.2 LSTM and Neuro-symbolic Models

We implemented a one-to-many LSTM using TensorFlow and Keras. For the pure (or vanilla) LSTM, we encode inputs as one-hot-encoded vectors representing the studentID, problemHierarchy and problemName. For the Neuro-symbolic model, we vectorize each instance, using a publicly available implementation of Obj2Vec [12] using the MLN formulas as specified in the previous section. Obj2Vec internally uses Gensim [28] to compute the embeddings for each symbol in the MLN. We use the embedding vectors of studentID, problemHierarchy and problemName as input to the LSTM encoder. Special Start and End tokens are used in the decoder section of the LSTM to identify the start and end of a prediction. The decoder unit predicts the KC at each time step, until an End token is found. To train the models in a feasible manner, we used a timeout of 3 hours. Within this timeout period, it was infeasible to use the all the instances since the training for the LSTM did not converge. Therefore, we randomly sampled instances to train our model within the specified limit. We refer to the trained models using random sampling as `LSTM-Random` and `LSTM-NS-Random` for the vanilla LSTM and the Neuro-symbolic models respectively. We further implemented a stratified-sampling/group based training on students and problems. For sampling by student, we selected N students from the student pool and for each selected student, we sampled M problems solved by that student. For sampling by problems, we selected N problems from the problem pool and sampled M students who have solved those problems. By increasing values of M and N, we progressively increased the instances as we show later in the results section. We refer to this as `LSTM-NS-NaiveGroup`.

5.1.3 Adaptive Training

We implemented K-Means clustering to cluster the data based on the embeddings and sampled from these clusters. We implemented a non-adaptive training model as follows. We independently clustered the students and the problems to generate C_1 student clusters and C_2 problem clusters. We then sampled one student from each student cluster and one problem from each problem cluster nearest to the cluster centers to create a training set of at most $C_1 * C_2$ instances that effectively covers all instances in our training data. We increase the number of clusters progressively starting from 100 student clusters and 1000 problem clusters to increase the number of instances in training as we show later in our results. We refer to this approach as `LSTM-NS-Clustered`. For our proposed adaptive weighting approach, we sample each cluster according to an importance weight. In each iteration, we update the importance weight of a cluster based on predictions made on a randomly sampled set of instances that were not used in training from that cluster. Here, we used 100 student clusters and 1000 problem clusters and

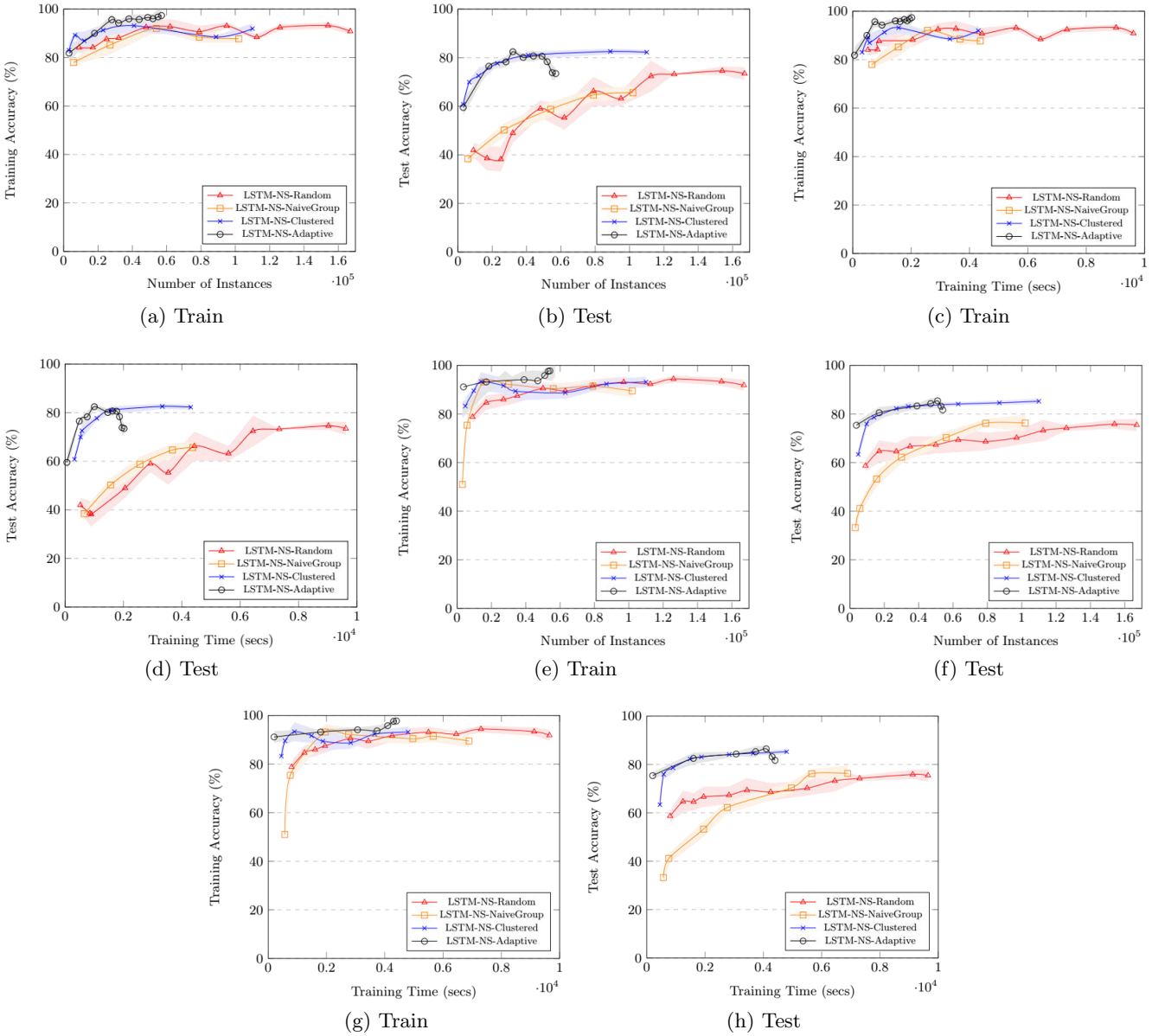

Figure 3: Accuracy results, the shaded portions show the standard deviation and the mean accuracy is plotted in the graphs. (a)-(d) corresponds to `Bridge to Algebra 2008, 2009` results and (e)-(h) corresponds to `Algebra 2008, 2009` results.

changed the importance weights of the clusters adaptively. We refer to this approach as `LSTM-NS-Adaptive`. The parameters for training our models are shown in Table 2.

5.2 Results

Fig. 3 compares the accuracy for different approaches. Fig. 3 (a), (b) shows the training and test accuracy respectively as we vary the number of instances used in training the models for the `Bridge to Algebra 2008, 2009` dataset. As we can see from these plots, `LSTM-NS-Adaptive` obtains the highest accuracy compared to the other methods. `HMM` (not shown in figures) gave us an accuracy of less than 5%. This low accuracy indicates that the strategy for diverse (or asymmetric) groups of students (or problems) cannot be represented by the same transition matrix. One possible approach that we will explore in future is to integrate our approach with HMMs, i.e., we learn an ensemble of HMMs where a HMM learns strategies for a symmetric group. A naive LSTM without embedding vectors, i.e., where inputs are a simple one-hot encoding of student and problems, also yields poor accuracy (less than 10%). This illustrates the importance of the latent features in the embedding vectors.

Note that we have only shown the results for the best performing model with latent dimension as 200 (experiments were carried out with different dimensions). As shown in Fig. 3 (a), (b), `LSTM-NS-Adaptive` and `LSTM-NS-Clustered` require a small fraction of the number of instances to obtain accuracy that is higher than `LSTM-NS-Random` and `LSTM-NS-NaiveGroup`. Further, the variance in accuracy of `LSTM-NS-Random` and `LSTM-NS-NaiveGroup` is much higher as shown by the shaded portion around the line plots compared to the variance of `LSTM-NS-Clustered` and `LSTM-NS-Adaptive`. `LSTM-NS-Adaptive` also obtains higher accuracy than `LSTM-NS-Clustered` as we adapt the weighting. However, note that the test accuracy starts to dip after the accuracy hits a peak value for `LSTM-NS-Adaptive`. This is because the adaptive cluster weights may focus too strongly on certain clusters in the training data which causes the LSTM to overfit. Therefore, in practice, we can stop the adaptation based on a validation set. Further, we can clearly see that exploiting symmetries in training leads to better generalization in Fig. 3 (b) where we see a significant difference between the accuracy for `LSTM-NS-Adaptive` and `LSTM-NS-Clustered` as compared to `LSTM-NS-NaiveGroup` and `LSTM-NS-Random` at smaller training sizes. Fig. 3 (c) and (d) also show the using relational symmetries in training results in a significant improvement in scalability since it shows that we can train `LSTM-NS-Adaptive` and `LSTM-NS-Clustered` in around half an hour to achieve an accuracy that is higher than the accuracy we obtain even after around 3 hours of training time in approaches where we do not choose training examples based on symmetries.

The results for the `Algebra 2008, 2009` dataset are similar to the ones for `Bridge to Algebra 2008, 2009`. As seen in Fig. 3 (e) and (f), the `LSTM-NS-Adaptive` model is the best performing model in terms of accuracy and it uses a small number of training instances to achieve this accuracy. Similar to the previous results, the variance for `LSTM-NS-NaiveGroup` and `LSTM-NS-Random` is much larger than that for `LSTM-NS-Adaptive` and `LSTM-NS-Clustered`. The training time shown in Fig. 3 (g) and (h) follow a similar pattern where `LSTM-NS-Adaptive` and `LSTM-NS-Clustered` can achieve high accuracy scores even with short training times since they take advantage of relational structure in the data.

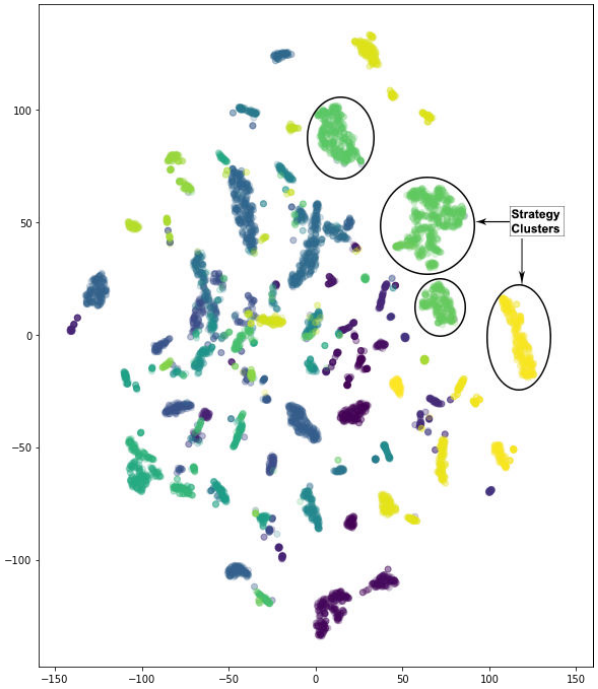

Figure 4: T-SNE visualization of strategies. The hidden layer of the final step in the LSTM is visualized for 100 test problems over all students. T-SNE reduces the latent LSTM vector to 2-D for visualization. Data points close together correspond to approximately similar strategies.

Table 3 shows the accuracy of predicting strategies for different problem units. Specifically, each problem in the dataset corresponds to a specific unit and we evaluate the models by testing the trained model on problems specific to a unit. For lack of space, we have not provided an exhaustive set of results for all units since there were around 50 units in the dataset. Instead, we provide accuracy results for the 10 units with largest number of data instances from the `Bridge to Algebra 2008, 2009` dataset. We see that on majority of the units, `LSTM-NS-Adaptive` has the best accuracy score. `LSTM-NS-Clustered` is the next best performing method. The difference in accuracy between units was significant in some cases. For instance, `LSTM-NS-Adaptive` had a very high accuracy for the unit `PERCENT CONVERSION` but a much lower accuracy for `ONE-STEP-EQUATIONS` and `TWO-STEP-EQUATIONS`. This may be due to higher complexity involved in solving equations as compared to problems involving percent conversion which may add to uncertainty in predicting strategies.

5.2.1 Structure in Strategies.

Finally, Fig. 4 shows a visualization of the strategies through a *T-SNE* plot. Specifically, we wanted to analyze if there are true patterns in the strategies. To do this, we use the `LSTM-NS-Adaptive` model to predict the strategy for 100 problems across all students. We show the results for `Bridge to Al-`

Unit	LSTM-NS-Random (%)	LSTM-NS-NaiveGroup (%)	LSTM-NS-Clustered (%)	LSTM-NS-Adaptive (%)
PROBABILITY	67.7	53.48	**77.45**	70.73
FRACTION-OPERATIONS-1	74.95	78.23	84.02	**90.49**
PERCENT-CONVERSION	99.01	99.18	88.6	**99.5**
SCI-NOTATION	66.67	69.39	69.9	**70.16**
PICTURE-ALGEBRA-2	90.23	**96.17**	93.29	95.17
RATIONAL-NUMBER-OPERATIONS	30.76	41.85	40.28	**66.13**
INTEGERS	87.47	78.83	93.52	**96.78**
ORDER-OF-OPERATIONS	60.03	50.35	45.53	**69.43**
ONE-STEP-EQUATIONS	**66.6**	53.36	58.86	58.06
TWO-STEP-EQUATIONS	**63.13**	63.03	59.42	61.5

Table 3: Comparing accuracy on test sets corresponding to different units in `Bridge to Algebra 2008, 2009`.

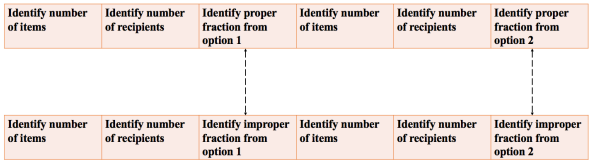

Figure 5: Strategies for different problems that have similar vector representations.

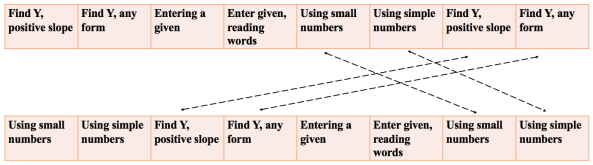

Figure 6: Strategies for the same problem that have similar vector representations.

`gebra 2008, 2009`. We use the hidden-vector from the last step of the LSTM as a representation of the strategy for a specific input instance. That is, this vector encodes information summarizing all the steps (or the full strategy) that the student performs when solving the input instance. We then plot this using the *T-SNE* plot that reduces the high-dimensional representation to a 2-D representation. Fig. 4 shows that there is a separation of different groups of strategies. The presence of such clusters of strategies indicates that there are indeed structures in student strategies and the representation learned by `LSTM-NS-Adaptive` discovers these symmetric structures.

Fig. 5 and 6 show two examples where the vectors representing the strategies are close to each other. Fig. 5 shows a case where the two strategies correspond to two different problems, one of which is related to proper fractions and the other to improper fractions. However, at a high level, these strategies are similar which is reflected in their similar vector representations. Another example shown in Fig. 6 shows a case where the two partial strategies shown correspond to the same problem are inversions of each other. That is, some of the steps in the two cases are performed in opposite orders. However, at a high-level, the strategies have symmetry which is reflected in their vector representations.

6. CONCLUSION

Predicting student strategies in problem solving can make AISs more engaging to students since the system can adapt itself to suit the student's strategy. In this paper, we described a Machine Learning approach to predict student strategies from large scale, structured student interaction data. Specifically, we adopted a Neuro-Symbolic approach, i.e., we combined LSTMs with a relational symbolic model to perform learning more efficiently. To do this, we encoded relationships in the data in the language of Markov Logic and based on relational symmetries in the data, we picked training instances are diverse. Doing this allowed us to learn our model to recognize diverse strategies at a smaller computational cost. Our evaluation on the KDD EDM challenge datasets show that our approach generalizes better and has significantly smaller training times as compared to approaches that do not exploit relational symmetries during learning. In future, we will extend our approach to datasets with finer-grained learner information and also develop joint inference models connecting mastery and strategies.

7. ACKNOWLEDGEMENTS

This research was sponsored by the National Science Foundation under the awards The Learner Data Institute (award #1934745) and NSF IIS award #2008812. The opinions, findings, and results are solely the authors' and do not reflect those of the funding agencies.

References

[1] G. Alain, A. Lamb, C. Sankar, A. C. Courville, and Y. Bengio. Variance reduction in SGD by distributed importance sampling. *CoRR*, abs/1511.06481, 2015.

[2] J. Austin. *How to Do Things with Words*. Oxford Press, 1962.

[3] M. Bannert, P. Reimann, and C. Sonnenberg. Process mining techniques for analysing patterns and strategies in students'self-regulated learning. *Metacognition and Learning*, 9(2):161–185, 2014.

[4] A. T. Corbett. Cognitive computer tutors: Solving the two-sigma problem. In *Proceedings of the 8th International Conference on User Modeling 2001*, page 137–147, 2001.

[5] E. D. Cubuk, B. Zoph, D. Mané, V. Vasudevan, and Q. V. Le. Autoaugment: Learning augmentation strategies from data. In *CVPR*, pages 113–123, 2019.

[6] P. T. Darga, M. H. Liffiton, K. A. Sakallah, and I. L. Markov. Exploiting structure in symmetry detection for cnf. In *Proceedings of the 41st Annual Design Automation Conference*, page 530–534, 2004.

[7] A. S. d'Avila Garcez, M. Gori, L. C. Lamb, L. Serafini, M. Spranger, and S. N. Tran. Neural-symbolic computing: An effective methodology for principled integration of machine learning and reasoning. *FLAP*, 6(4):611–632, 2019.

[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and F.-F. Li. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255. IEEE Computer Society, 2009.

[9] P. Domingos and D. Lowd. *Markov Logic: An Interface Layer for Artificial Intelligence*. Morgan & Claypool, San Rafael, CA, 2009.

[10] A. Grover and J. Leskovec. Node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, page 855–864, 2016.

[11] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[12] M. M. Islam, S. Sarkhel, and D. Venugopal. On lifted inference using neural embeddings. In *AAAI Conference on Artificial Intelligence*, pages 7916–7923, 2019.

[13] T. B. Johnson and C. Guestrin. Training deep models faster with robust, approximate importance sampling. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, page 7276–7286, 2018.

[14] A. Katharopoulos and F. Fleuret. Not all samples are created equal: Deep learning with importance sampling. In *Proceedings of the 35th International Conference on Machine Learning*, pages 2525–2534, 2018.

[15] T. Khot, N. Balasubramanian, E. Gribkoff, A. Sabharwal, P. Clark, and O. Etzioni. Exploring markov logic networks for question answering. In *EMNLP*, pages 685–694, 2015.

[16] J. S. Kinnebrew and G. Biswas. Identifying learning behaviors by contextualizing differential sequence mining with action features and performance evolution. In *International Conference on Educational Data Mining*, pages 57–64, 2012.

[17] J. S. Kinnebrew, K. Loretz, and G. Biswas. A contextualized, differential sequence mining method to derive students' learning behavior patterns. *Journal of Educational Data Mining*, 5(1):190–219, 2013.

[18] J. S. Kinnebrew, J. R. Segedy, and G. Biswas. Integrating model-driven and data-driven techniques for analyzing learning behaviors in open-ended learning environments. *IEEE Trans. Learn. Technol.*, 10(2):140–153, 2017.

[19] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017*, 2017.

[20] K. R. Koedinger, A. T. Corbett, and C. Perfetti. The knowledge-learning-instruction framework: Bridging the science-practice chasm to enhance robust student learning. *Cogn. Sci.*, 36(5):757–798, 2012.

[21] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[22] L. C. Lamb, A. S. d'Avila Garcez, M. Gori, M. O. R. Prates, P. H. C. Avelar, and M. Y. Vardi. Graph neural networks meet neural-symbolic computing: A survey and perspective. In *IJCAI*, pages 4877–4884, 2020.

[23] K. Leelawong and G. Biswas. Designing learning by teaching agents: The betty's brain system. *Int. J. Artif. Intell. Ed.*, 18(3):181–208, Aug. 2008.

[24] N. Maharjan, D. Gautam, and V. Rus. Assessing free student answers in tutorial dialogues using LSTM models. In *Artificial Intelligence in Education - 19th International Conference, AIED*, pages 193–198. Springer, 2018.

[25] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Neural Information Processing Systems*, pages 3111–3119, 2013.

[26] D. M. Morrison, B. Nye, V. Rus, S. Snyder, J. Boller, and K. B. Miller. Tutorial dialogue modes in a large corpus of online tutoring transcripts. In *Artificial Intelligence in Education - 17th International Conference*, volume 9112, pages 722–725. Springer, 2015.

[27] M. Niepert. Markov chains on orbits of permutation groups. In *UAI*, pages 624–633. AUAI Press, 2012.

[28] R. Řehůřek and P. Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, 2010.

[29] S. Ritter. Communication, cooperation and competition among multiple tutor agents. In *Artificial Intelligence in Education: Knowledge and media in learning systems*, pages 31–38, 1997.

[30] S. Ritter, R. Baker, V. Rus, and G. Biswas. Identifying strategies in student problem solving. *Design Recommendations for Intelligent Tutoring Systems*, 7:59–70, 2019.

[31] V. Rus, S. K. D'Mello, X. Hu, and A. C. Graesser. Recent advances in conversational intelligent tutoring systems. *AI Magazine*, 34(3):42–54, 2013.

[32] V. Rus, N. Maharjan, L. J. Tamang, M. Yudelson, S. R. Berman, S. E. Fancsali, and S. Ritter. An analysis of human tutors' actions in tutorial dialogues. In *Proceedings of the Thirtieth International Florida Artificial Intelligence Research Society Conference, FLAIRS*, pages 122–127, 2017.

[33] J. R. Searle. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, 1969.

[34] J. Stamper, A. Niculescu-Mizil, S. Ritter, G. Gordon, and K. Koedinger. Algebra I 2008-2009. Challenge data set from KDD Cup 2010 Educational Data Mining Challenge. Technical report, 2010.

[35] R. Stewart and S. Ermon. Label-free supervision of neural networks with physics and domain knowledge. In S. P. Singh and S. Markovitch, editors, *AAAI*, pages 2576–2582, 2017.

[36] D. Venugopal and V. Rus. Joint inference for mode identification in tutorial dialogues. In *COLING 2016, 26th International Conference on Computational Linguistics*, pages 2000–2011. ACL, 2016.

[37] J. Wong, M. Khalil, M. Baars, B. D. Koning, and F. Paas. Exploring sequences of learner activities in relation to self-regulated learning in a massive open online course. *Computers & Education*, 140:103595, 2019.

[38] K. Yi, J. Wu, C. Gan, A. Torralba, P. Kohli, and J. B. Tenenbaum. Neural-Symbolic VQA: Disentangling Reasoning from Vision and Language Understanding. In *Advances in Neural Information Processing Systems (NIPS)*, 2018.

[39] Y. You, J. Hseu, C. Ying, J. Demmel, K. Keutzer, and C.-J. Hsieh. Large-batch training for lstm and beyond. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019.